Stuck Detection

Tai2.tran

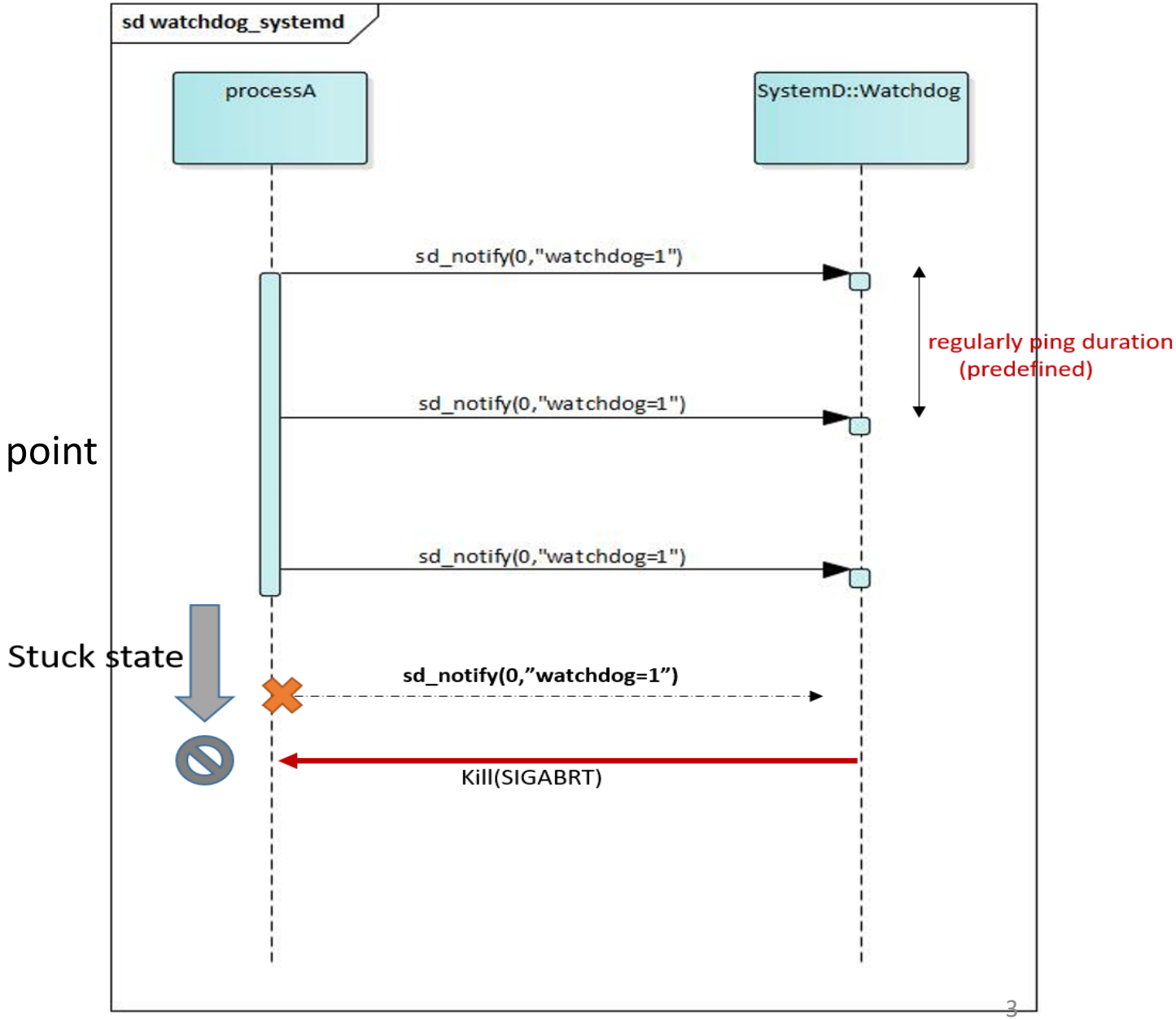
Background

- Detect stuck at process in run time
  - Systemd supports watchdog library to check health of processes on Linux
  - Watchdog library on tiger platform supports detect stuck point on binder function
  - None of library help developer to find exact stuck point.

- Motivation
  - Design a module to support finding exact stuck point (without exact stuck point, developer spend much effort by days to debug)
  - The module should light-weight and can be applied to every Linux platform
  - The module should support multiple concurrent clients for tracking stuck
  - It also supports calculating elapsed time of any given API on clients

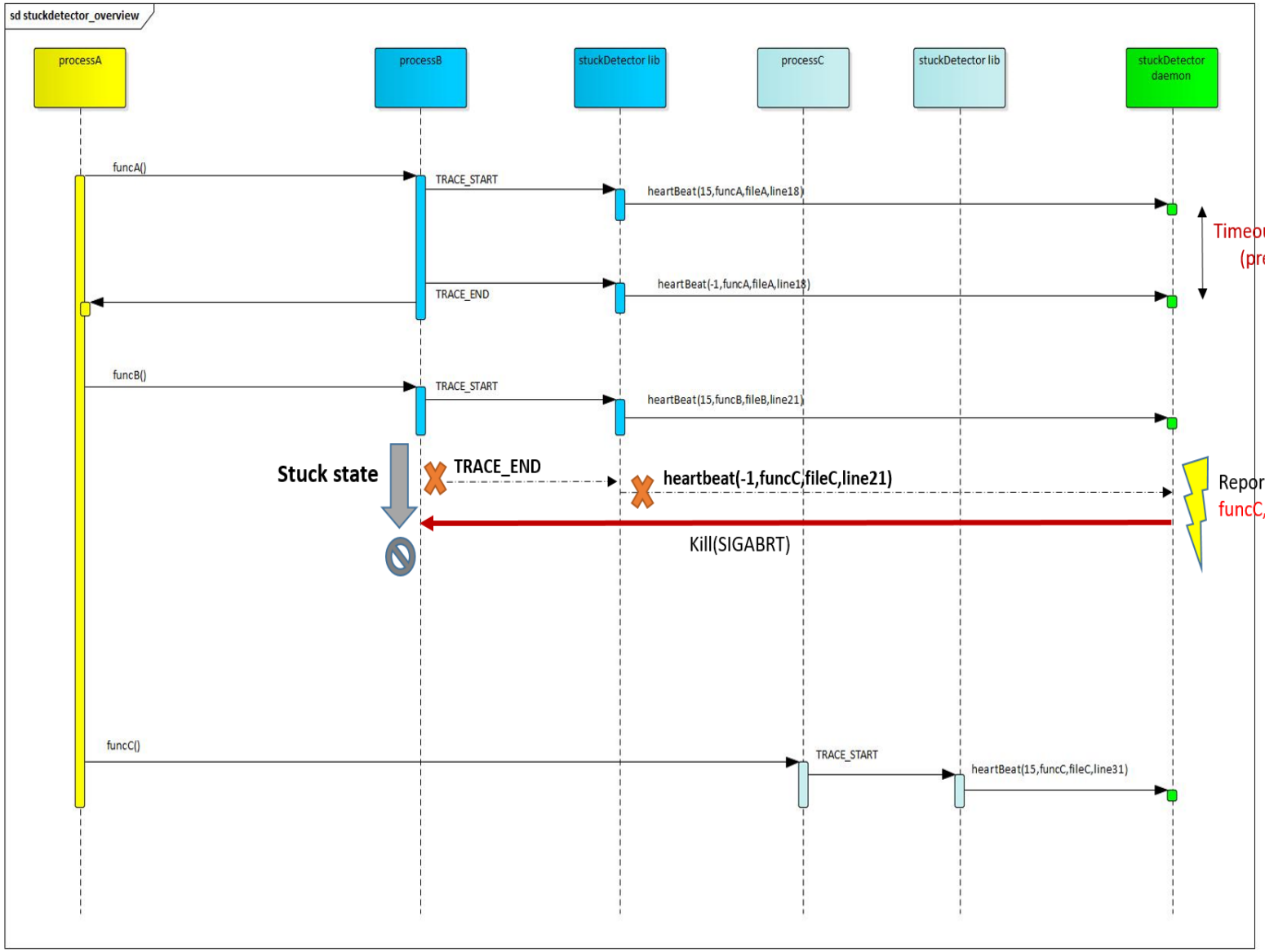
Background

- Watchdog of systemD
  - Client pings heartbeat periodically to SystemD
  - SystemD can detect client process crash but not showing exactly crash point



Background:
Stuck detector overview

- Show exactly stuck point location
- Support multiple client connection simultaneously
- Don’t block client connection
  - Tiger watchdog lib can block client



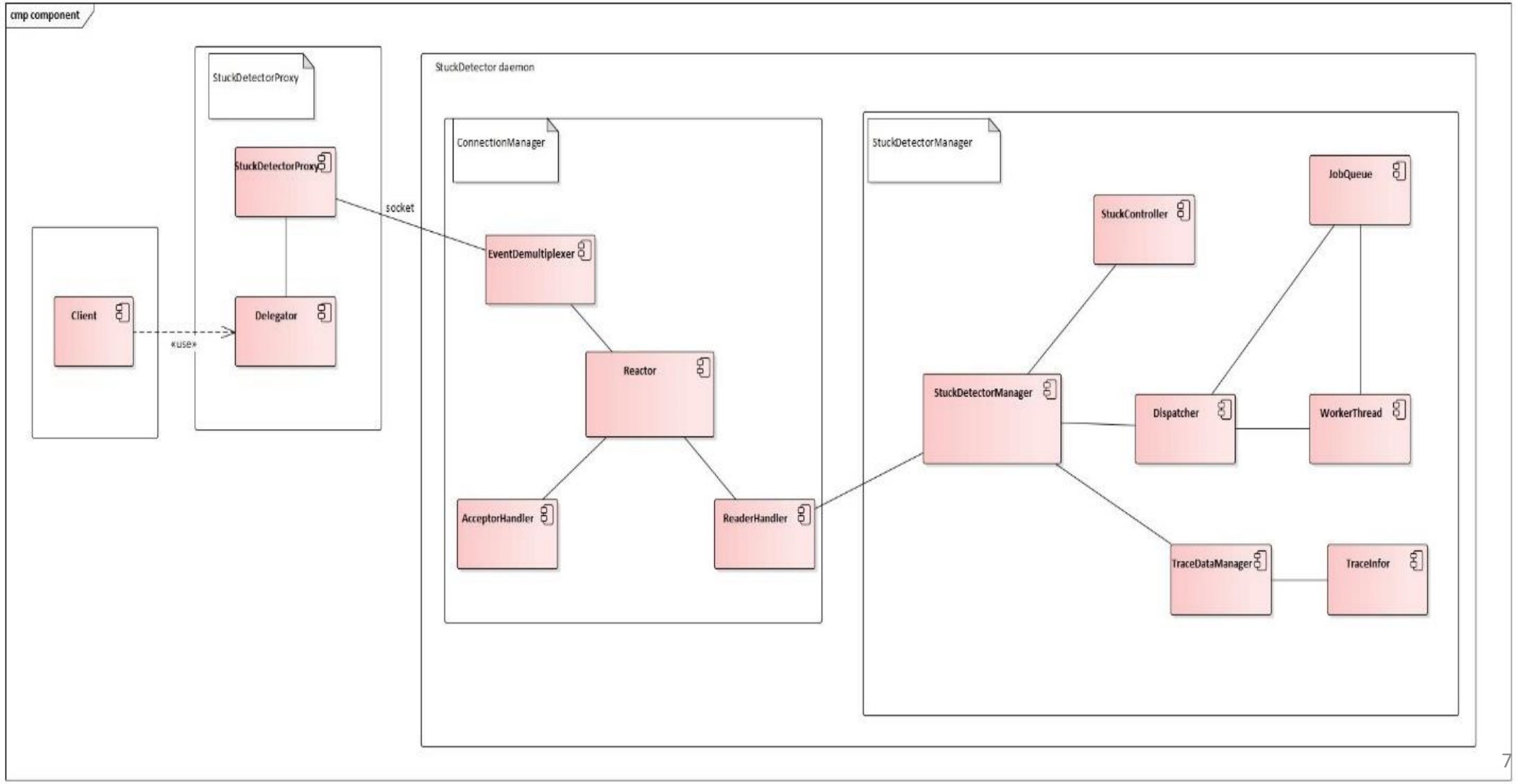
Features

- Manage connection at client side
- Manage connection at server side
- Handle multiple concurrent requests asynchronously
- Algorithm to detect stuck point and measure elapsed time of client's API
- Manage data base to store tracer information

Quality attributes

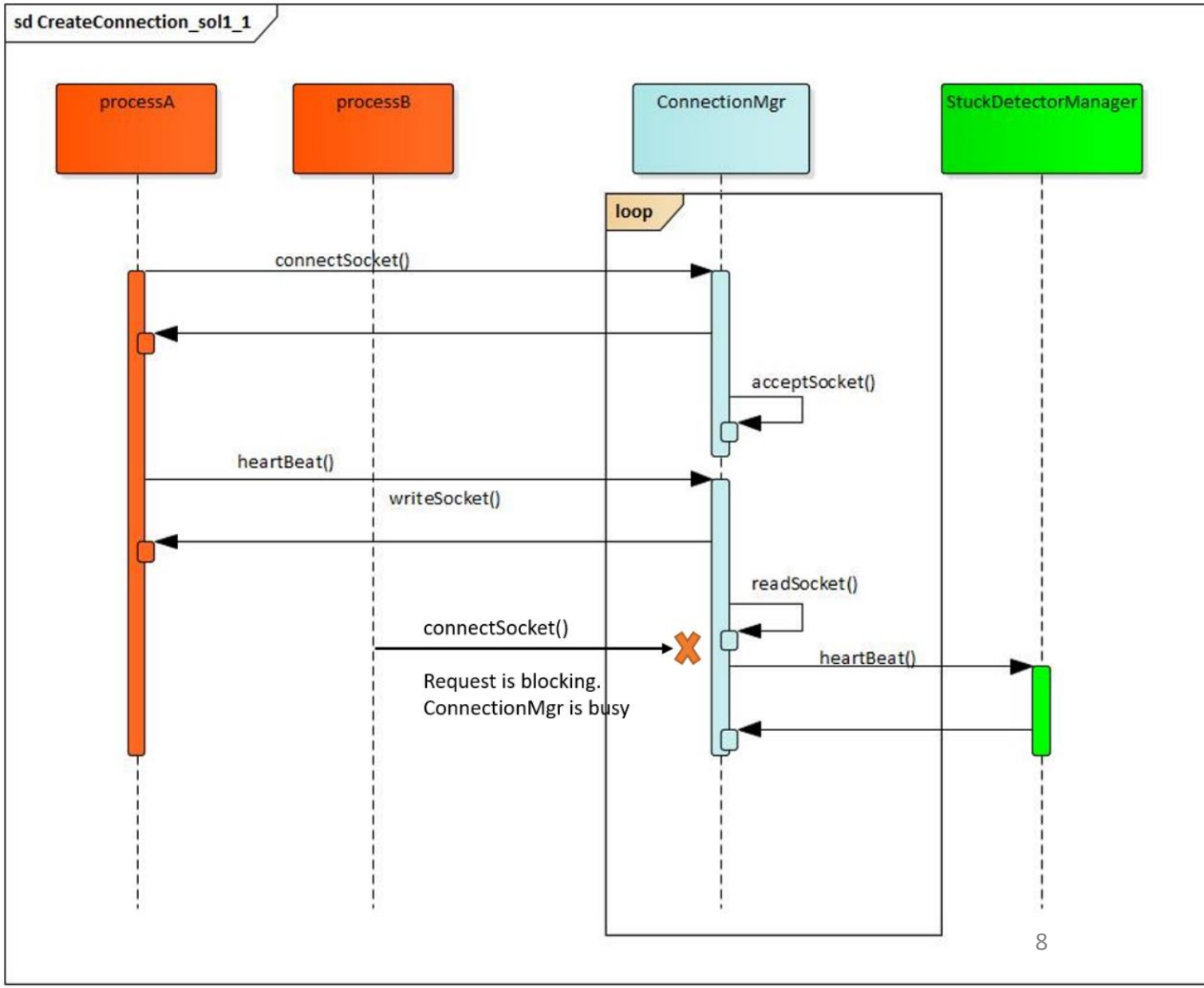
Scenario #	QA scenario	Quality attribute	priority
1	Support multiple concurrent requests without blocking client request	Performance	High
2	Support simple API for client	Reusability	Mid
3	Detect correct stuck point	Reliability	High
4	Deploy on every Linux platform	Reusability	Mid

Software Architectural design



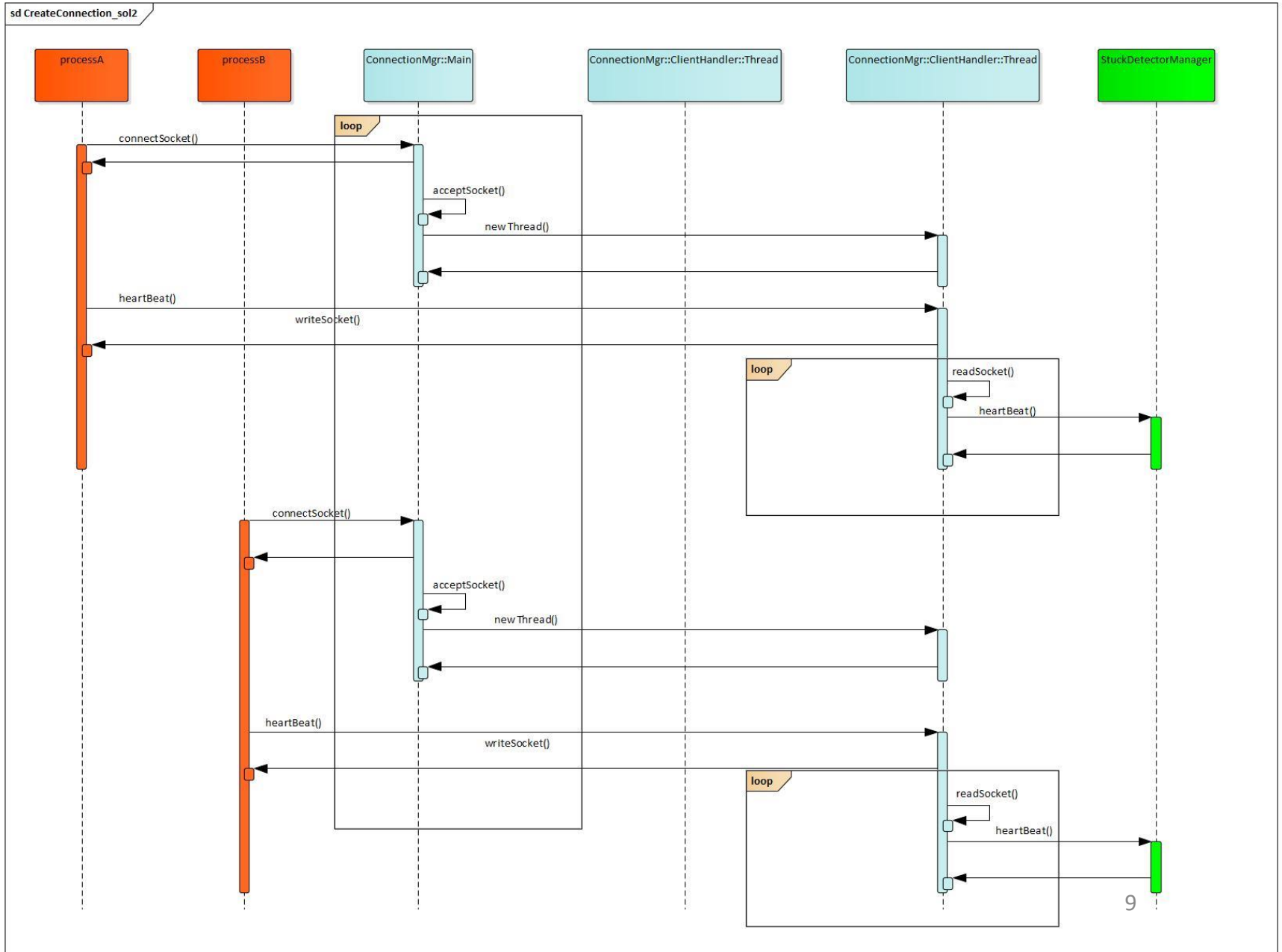
ConnectionMgr design 1

- A looper to wait connection request or heartbeat request form clients
- Drawback: Other clients are being blocked when ConnectionMgr is busy to handle current client



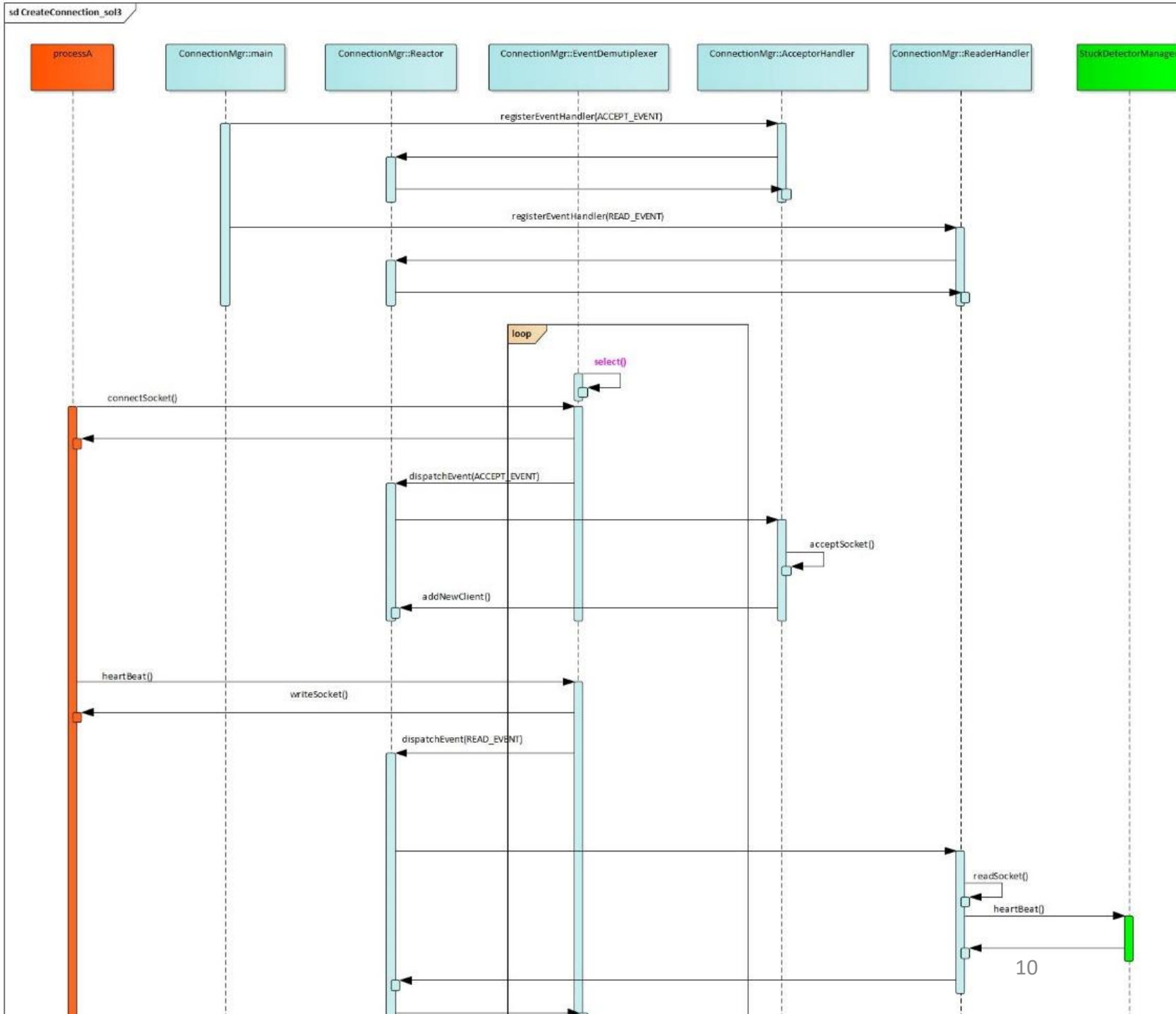
ConnectionMgr design 2

- Support concurrent clients by multithreading.
- Drawback:
  - Inefficient and non-scalable due to context switching, synchronization and data movement among CPUs.
  - Take high resource

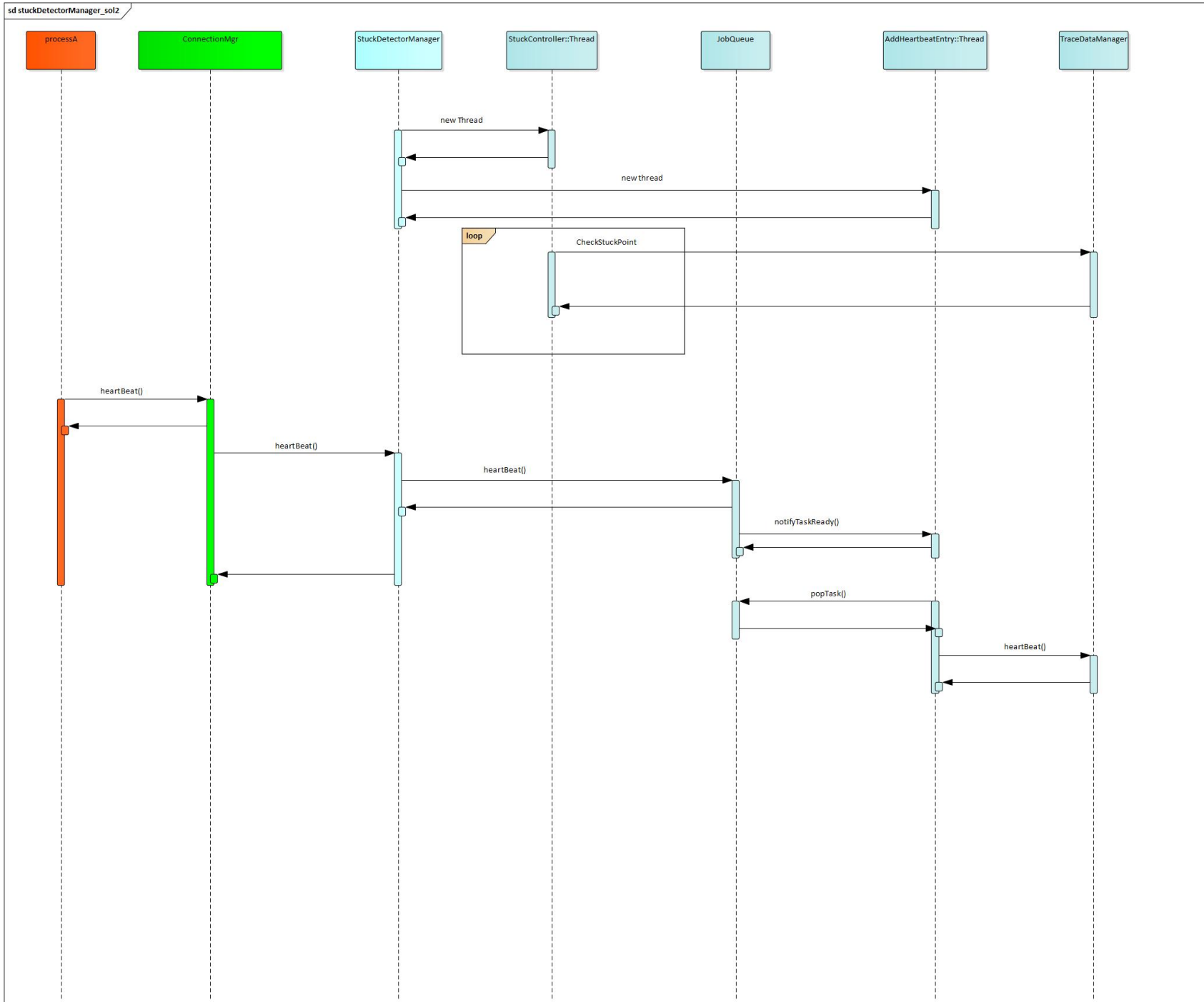


ConnectionMgr design 3 (selected method)

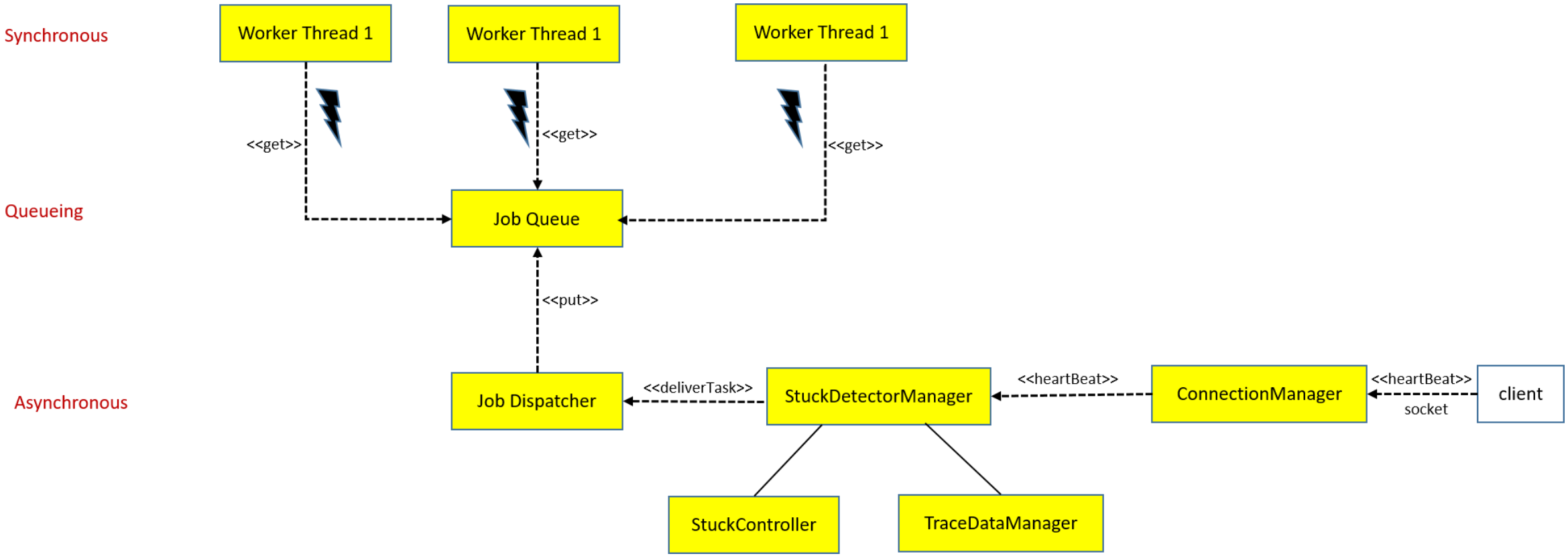
- Using event-based model
- EventDemultiplexer synchronously wait for arrival of events.
- Then notify to associated Hander
- Using I/O multiplexing for wait events



StuckdetectorMgr design 1

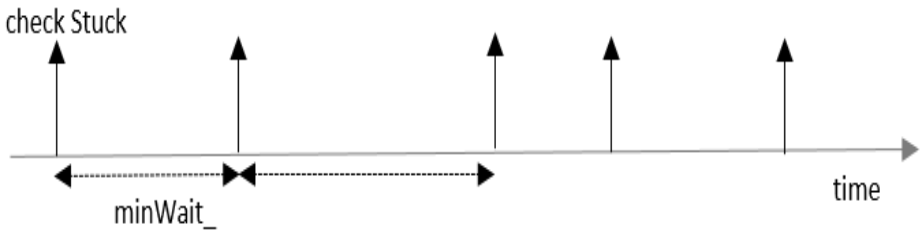


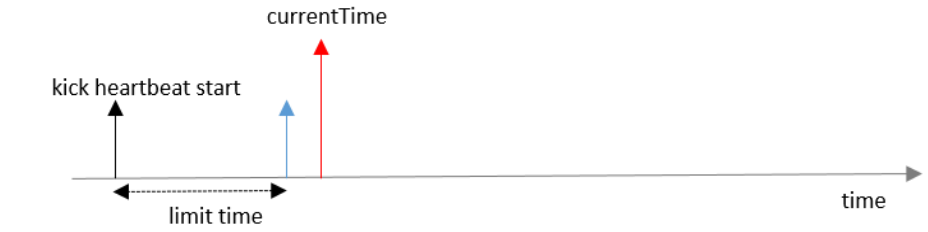
StuckdetectorMgr design 2



Algorithm design

- Main Looper check stuck
  - after a minwait timeout, looper check stuck
  - scan each entry in trace data, a stuck on an entry if:
    - timeout > 0 and currentTime > limitTime

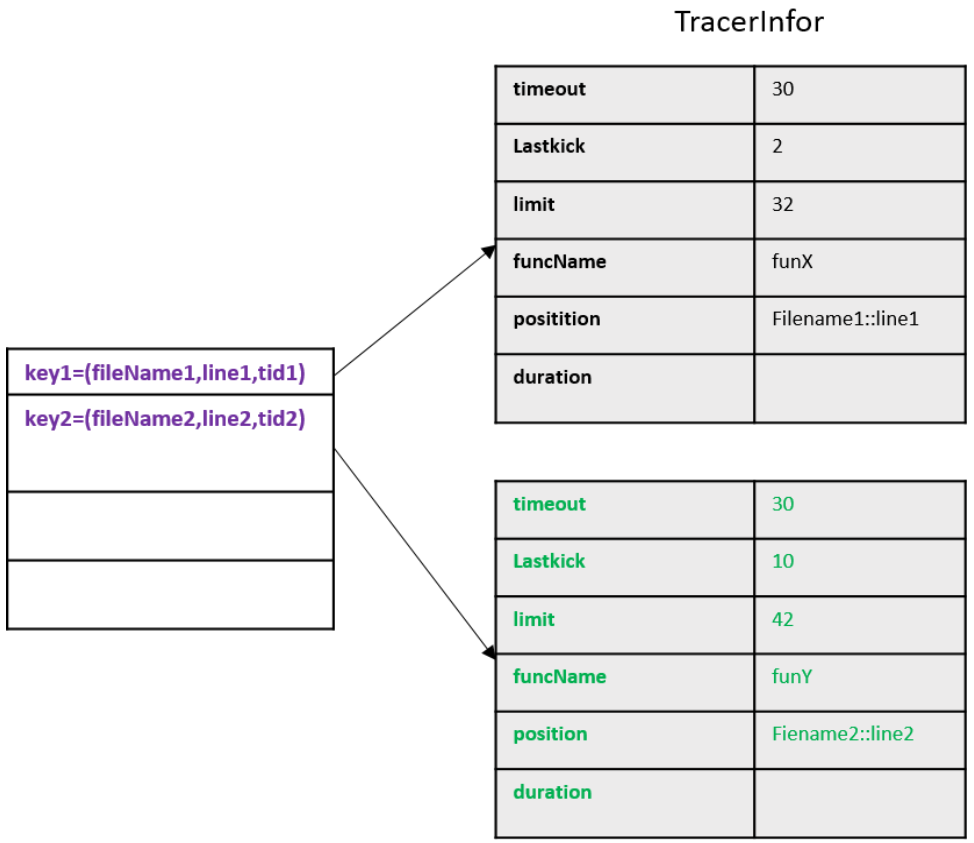




client must kick heartbeat end during limit time

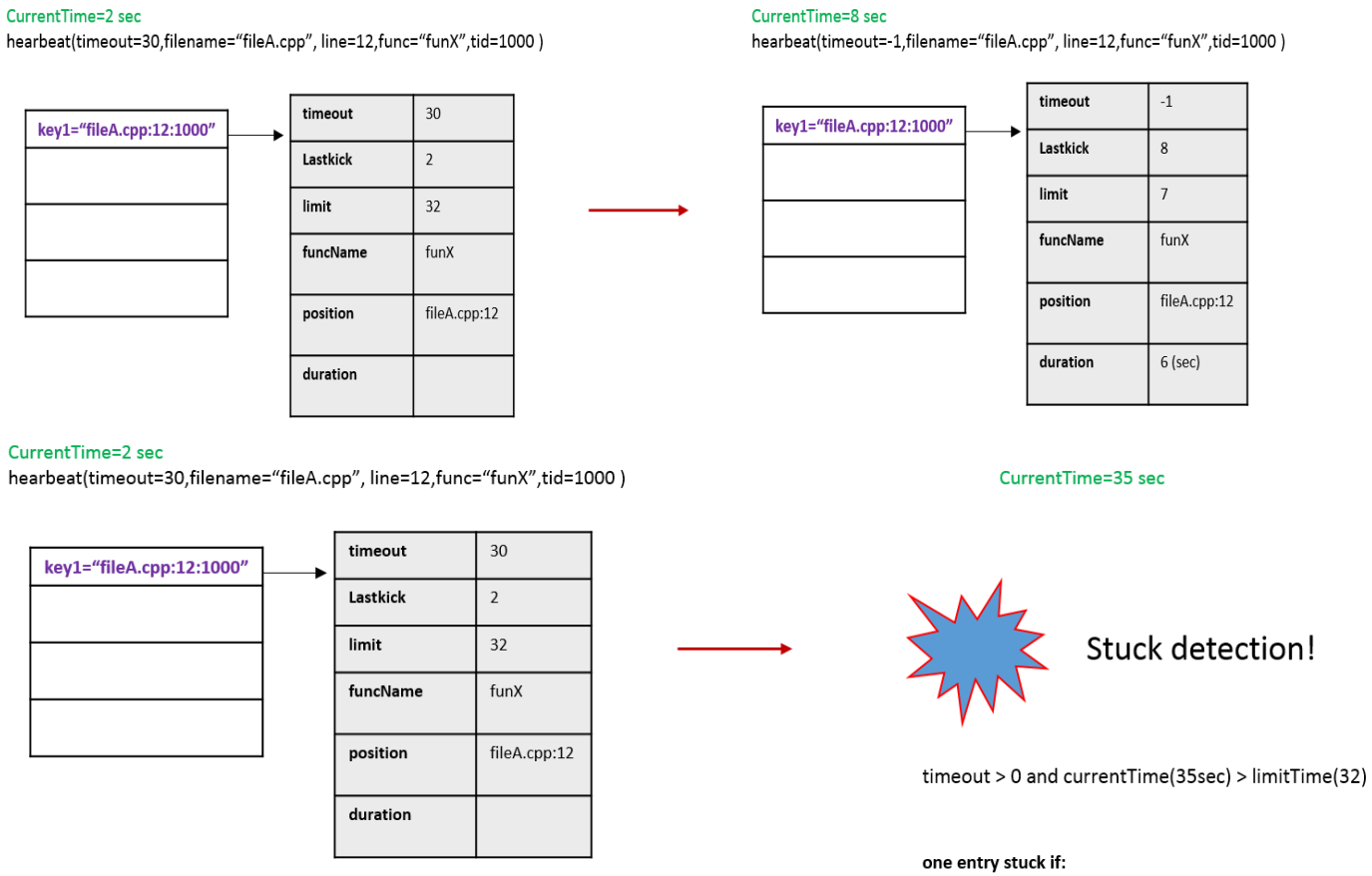
Algorithm design

- Data structure to store trace information

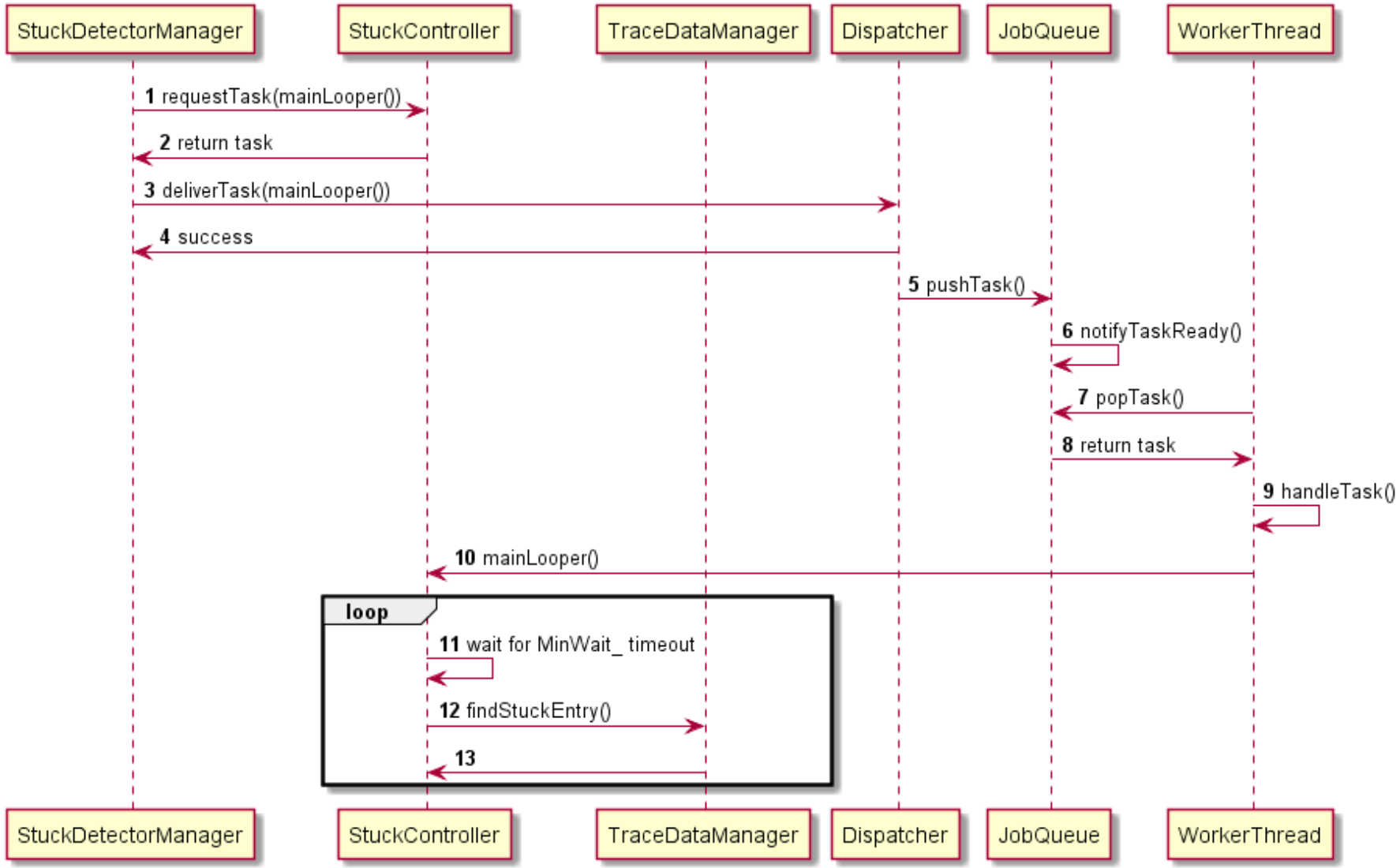


Algorithm design

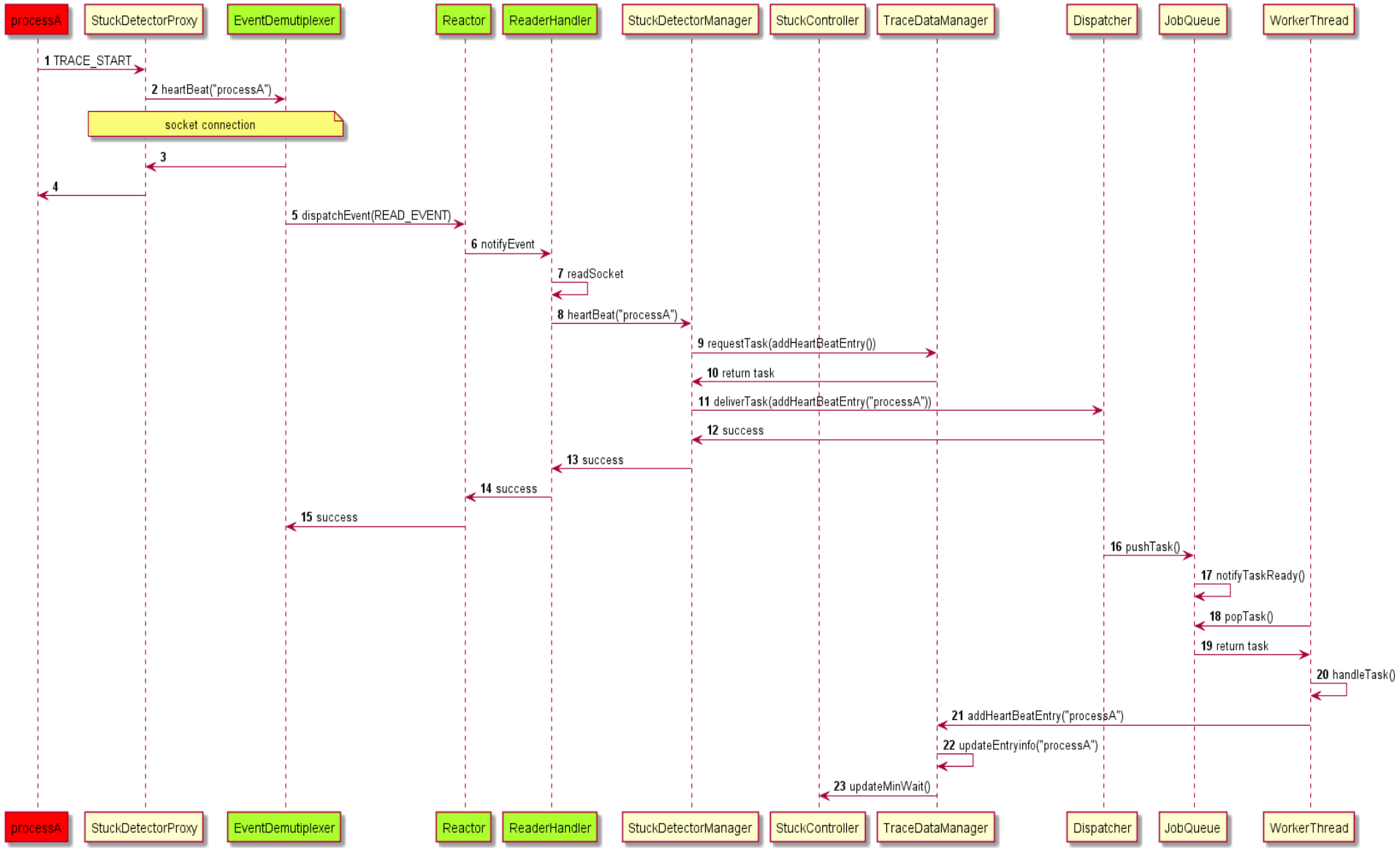
- Update trace information



Start main looper to detect stuck

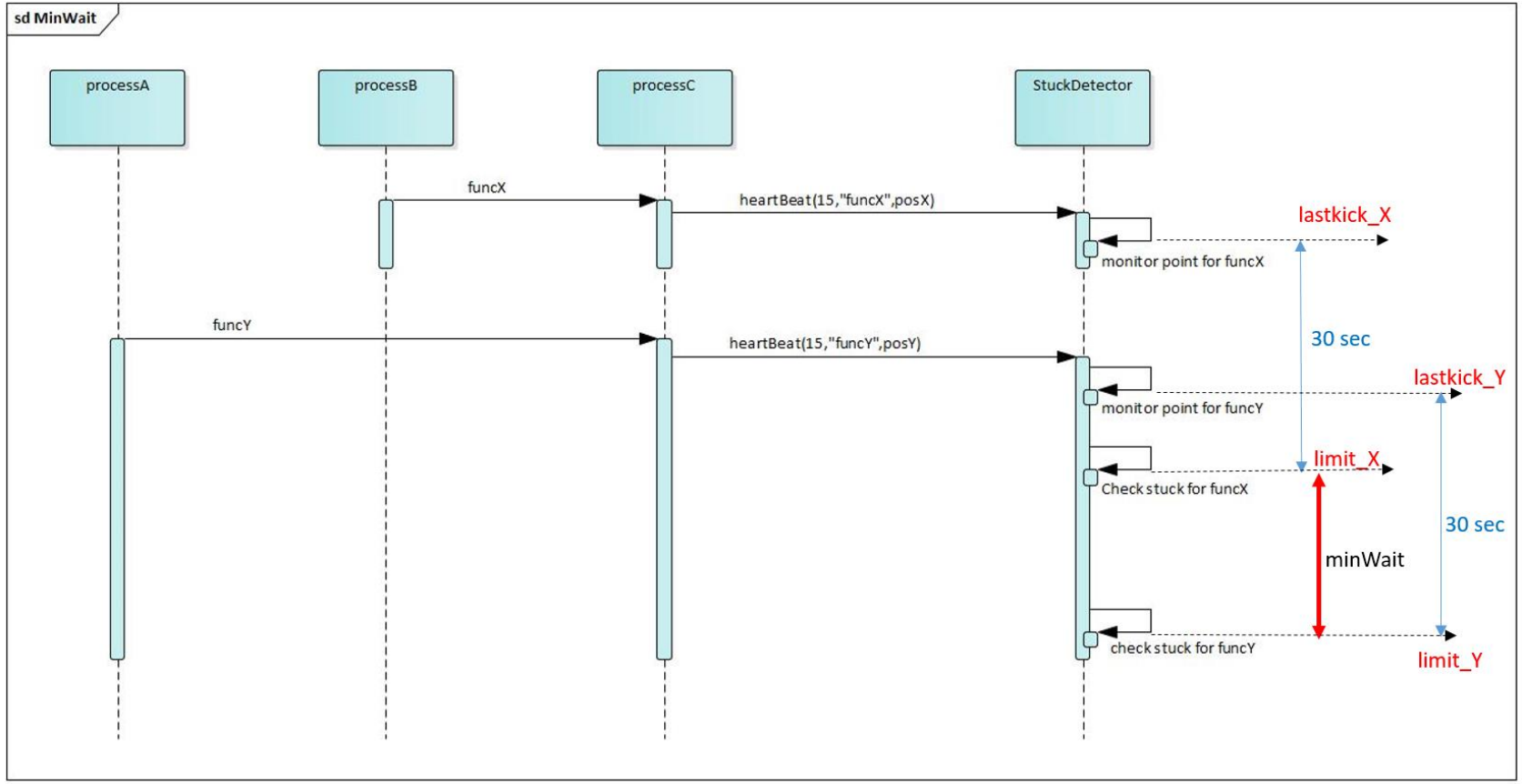


Client kick heartbeat



Algorithm design

- Update minWait time
  - minWait = minimum{limitTime[i]} – currentTime();



Demo result

- Setup test environment
  - Stuckdector select default timeout value is 15 sec
  - Module ADADiagMgr implements 2 functions
    - First function:, simulate readDataBlock() with a delay of 9 sec
    - Second function: clearDiagInfo(), simulate with a delay of 30 sec
  - Run first function:
    - slddgm ada send_request 0 0x97 0x0122 0 0 0 0 0x48 0xB8 0x00
  - Run second function
    - slddgm ada send_request 1 0x97 0x114 0 0 0 0
- Expected result:
  - Stuckdetector shows elapsed time of first function: ~9 sec
  - Stuckdetector detects stuck on second function, then abort ADADiagMgr process

Demo result: setup environment

Add TRACE_START

Simulate Delay

```
00054: void ADADiagManagerService::ADAMgrOperator::readDataBlock(const android::sp<ADA_request>& req)
00055: {
00056:     TRACE_START;
00057:
00058:     // This is functional request for dacsi $0122, read DID for single DID block
00059:     // Backend request data
00060:     // byte[0]=did MSB, byte[1]=did LSB, byte[2]=enable PN
00061:
00062:     // Make request to DiagMgr
00063:     // byte[0]=0x22, byte[1]=did MSB, byte[2]=did LSB
00064:
00065:     const char *dacsi_name="read data block by identifier";
00066:     std::string msg;
00067:
00068:     LOGI("handle readDataBlock");
00069:
00070:     std::vector<std::uint8_t> mADABytes = req->getADAbytes();
00071:     if (mADABytes.size() < 3) {
00072:         msg.assign("invalid request");
00073: #ifndef __UNITTEST__
00074:         throw ADAException(msg);
00075: #endif
00076:
00077:     }
00078:
00079:     sleep(9);
00080:
00081:     uint16_t target_value = static_cast<uint16_t>(req->getTargetValue());
00082:     uint16_t dacsi = req->getDACSI();
00083:
00084:     LOGI("mode=%d, targetID=0x%02X, dacsi=0x%04X",(int)req->getMode(),target_value, dacsi);
00085:
```

Demo result: setup environment

Add TRACE_START

Simulate Delay

```
00024: // Functional request: $0114
00025: void ADADiagManagerService::ADAMgrOperator::clearDiagInfo(const android::sp<ADA_request>& req)
00026: {
00027:     TRACE_START;
00028:
00029:     const char *dacsi_name="Clear Diag Info";
00030:     std::string msg;
00031:
00032:     if (mParent.mADADiagInputMgr->mRemoteDiagMgr != nullptr) {
00033:         LOGI("send clearDiagnosticInformation");
00034:         checkMode(static_cast<uint8_t>(req->getMode()));
00035:         LOGI("data from GMS should empty for $0114");
00036:
00037:         if (req->getMode() == pal::ada::target_id_mode::functional) {
00038:             LOGI("this is functional request");
00039:             LOGI("send request to internal TCP");
00040:             mParent.mADADiagInputMgr->mRemoteDiagMgr->clearDiagnosticInformation(MODULE_ADA,4,EDIAG_COM_INTERNAL,0,ml
00041:             LOGI("send request to external ECUs");
00042:             mParent.mADADiagInputMgr->mRemoteDiagMgr->clearDiagnosticInformation(MODULE_ADA,4,EDIAG_COM_CAN,0,mParen
00043:         }
00044:     } else {
00045:         msg.assign("mRemoteDiagMgr is null");
00046: #ifndef __UNITTEST__
00047:         throw ADAException(msg);
00048: #endif
00049:     }
00050:
00051:     sleep(30);
00052: } « end clearDiagInfo »
00053:
```

Demo result

- Stuckdetect shows elapsed time of first function.

```
[ StuckController : 745 : 782 ] exit waiting
[ StuckController : 745 : 782 ] timeout occur, exit waiting
[ StuckController : 745 : 782 ] check stuck
[ TraceDataManager : 745 : 782 ] [stuckDetector] current_time=443 -------
[ TraceDataManager : 745 : 782 ] released client,tid=767,pos=ADADiagHandleRequest.cpp:56,funcName_=readDataBlock,timeout_sec=-1
[ TraceDataManager : 745 : 782 ] lastkick=437,Limit=436, duration=9003 ms
[ StuckController : 745 : 782 ] update min wait
[ StuckController : 745 : 782 ] [watchdog] main looper is running
[ StuckController : 745 : 782 ] start waiting: min_wait=15 sec
[ StuckController : 745 : 782 ] exit waiting
[ StuckController : 745 : 782 ] timeout occur, exit waiting
[ StuckController : 745 : 782 ] check stuck
[ TraceDataManager : 745 : 782 ] [stuckDetector] current_time=458 -------
[ StuckController : 745 : 782 ] update min wait
[ StuckController : 745 : 782 ] [watchdog] main looper is running
[ StuckController : 745 : 782 ] start waiting: min_wait=15 sec
```

Demo result

- Stuckdetect detects stuck at second function, then abort the client process